Generating Difficult CNF Instances in Unexplored Constrainedness Regions

GUILLAUME ESCAMOCHER, BARRY O'SULLIVAN, and STEVEN DAVID PRESTWICH,
Insight Centre for Data Analytics, School of Computer Science & IT, University College Cork, Ireland

When creating benchmarks for SAT solvers, we need CNF instances that are easy to build but hard to solve. A recent development in the search for such methods has led to the Balanced SAT algorithm, which can create k-CNF instances with m clauses of high difficulty, for arbitrary k and m. In this paper we introduce the No-Triangle CNF algorithm, a CNF instance generator based on the cluster coefficient graph statistic. We empirically compare the two algorithms by fixing the arity and the number of variables, but varying the number of clauses. We find that the hardest instances produced by each method belong to different constrainedness regions. In the 3-CNF case for example, hard No-Triangle CNF instances reside in the highly-constrained region (many clauses), while Balanced SAT instances obtained from the same parameters are easy to solve. This allows us to generate difficult instances where existing algorithms fail to do so.

ACM Reference format:
Guillaume Escamocher, Barry O'Sullivan, and Steven David Prestwich. 2020. Generating Difficult CNF Instances in Unexplored Constrainedness Regions. *ACM J. Exp. Algor.* 1, 1, Article 1 (January 2020), 12 pages.
https://doi.org/10.1145/3385651

1 INTRODUCTION

The Boolean Satisfiability Problem, commonly called SAT, is the problem of assigning values to a set of variables while satisfying a set of disjunctive clauses, each consisting of literals representing either a variable or its negation. An instance of this problem is called a CNF instance when represented in Conjunctive Normal Form. Results on the hardness of random CNF instances have the potential to positively impact the performance and effectiveness of solvers in many different decision and optimization domains, from Artificial Intelligence to engineering applications [2]. This has led to significant effort being devoted to find efficient ways to generate random CNF instances that can be customized by tuning pertinent parameters.

State-of-the-art SAT solvers perform extremely well on industrial CNF instances of large dimensions. There exist however families of small CNF instances with explicitly defined interdependent parameters (such as the number of variables or the number of clauses) that are challenging for these same solvers. Therefore, in order to develop and test the new techniques for solving SAT, it is important to have a way to create at will difficult CNF instances with any combination of parameters. Encoding a different type of problem into SAT is not a practical way to achieve this goal, as the conversion to CNF would considerably alter the structure of the original instance and prevent precise control of the main parameters (number of literals in each clause, number of variables, and number of clauses).

1084-6654/2020/1-ART1 $15.00
https://doi.org/10.1145/3385651

In major SAT competitions [11], the smallest instances that cannot be solved within the time limit are obtained by hiding cardinality constraints. They offer a great deal of difficulty to solvers when presented for the first time, but can usually be solved by special techniques when identified. As an illustration, the sgen6 algorithm is able to generate instances with fewer than two hundred variables that no solver can solve in less than 10 minutes [20], but these instances can be reduced to a simple matching problem [14].

A more general drawback of this kind of generator is that they are not easily parameterizable. Because of their rigid structure, fixing one parameter (such as the number of variables) constrains the other parameters (such as the arity or the number of clauses) to very specific values. The sgen6 algorithm, for example, outputs instances with clauses of different arity, including clauses with only two literals, and with a fairly low number of clauses compared to the number of variables (approximately twice as many). Generators that rely on algorithm configuration tools have a larger degree of freedom, but still have at least one parameter that they do not control, usually the number of clauses [5]. Furthermore, using a configuration tool implies a costly preprocessing phase.

To address these issues, a new method of generating difficult instances has been proposed, called Balanced SAT [19]. The main idea of Balanced SAT is to balance the number of occurrences of each literal, as well as minimizing the number of variable pairs that appear in different clauses. Instances created by Balanced SAT do not challenge solvers as much as the hardest crafted instances, but because of their random nature there is no known special technique that can easily solve them. Furthermore, Balanced SAT can generate instances of any arity, any number of variables, and any number of clauses.

While the Balanced SAT algorithm can create extremely varied instances, not all of these are hard to solve. Indeed, satisfiability problems exhibit a phase transition phenomenon, in which instances under a given constrainedness (ratio of clauses to variables) threshold are increasingly likely to be satisfiable as the size increases, while instances with greater constrainedness are increasingly likely to be unsatisfiable [7]. Empirically, instances close to this threshold are found to be extremely hard, while under- and highly-constrained instances are found to be much easier [16].

For CNF instances with n variables, m clauses, and exactly 3 literals in each clause, the threshold is conjectured to lie between $m = 4.2n$ and $m = 4.3n$ (values of 4.258 [10] and 4.267 [15] have been advanced). For completely random instances this ratio is where the hard instances are. The peak of difficulty for instances built by generators of hard instances is distinctly lower. As mentioned above, the ratio for sgen6 instances is about 2, while for Balanced SAT it is approximately 3.6 [19]. These low figures might possibly be explained by the fact that instance generators of this kind try to force solvers to look at a large number of possible variable assignments, and adding more clauses decreases the number of paths to explore. In any case, we are not aware of any existing generator of difficult instances with an observed peak in the highly-constrained region of random 3-CNF instances.

Our main contribution in this paper is a new algorithm for generating CNF instances, called No-Triangle CNF. Like Balanced SAT, No-Triangle CNF can generate instances of any arity, any number of variables, and any number of clauses. It balances the number of occurrences of each literal, again like Balanced SAT, but as well as avoiding redundant variable pairs, it also minimizes the number of constraint triangles in the constraint graph associated with the instance. Constraint triangles, defined in Section 2.1, are related to the cluster coefficient measure of the constraint graph, the graph that indicates which pairs of variables appear in a same clause.

While parts of our No-Triangle algorithm are inspired from Balanced SAT, we show in Section 3 that the behavior of instances obtained from these two generators is drastically distinct. For a fixed arity and a fixed number of variables, difficult No-Triangle CNF instances appear at a different number of clauses than Balanced SAT instances. No-Triangle CNF instances that are hard to solve are generated from parameters from which Balanced SAT only gives easy instances. No-Triangle CNF thus constitutes a way to create difficult instances in constrainedness regions where they were not found before.

In the next section we describe both the existing Balanced SAT generator and our novel No-Triangle CNF generator. We also define some graph notions that are integral to our algorithm. In Section 3 we present the core results of the paper, empirical studies for different instance sizes and arities of the behavior of No-Triangle CNF instances compared to Balanced SAT as well as random instances. Finally we conclude in Section 4.

2 GENERATORS

2.1 Preliminary notions

We begin by formally defining the Boolean Satisfiability Problem (SAT).

Definition 2.1. A *CNF instance* is composed of n variables $v_1, v_2, \dots, v_n$ and m clauses $C_1, C_2, \dots, C_m$, where each clause C_i is a disjunction of k_i literals $l_1 \vee l_2 \vee \cdots \vee l_{k_i}$ and each literal is either a variable from $\{v_1, v_2, \dots, v_n\}$ or the negation of one such variable. The *arity* of the CNF instance is the number of literals in the clause with the most literals.

A literal is *positive* if it corresponds to a variable, and *negative* if it corresponds to the negation of a variable. The *polarity* of a literal is its sign. A *k-CNF instance* is a CNF instance with exactly k literals in each clause. A *solution* for a (k-)CNF instance I is an assignment of boolean values to all n variables such that all m clauses of I are satisfied.

The problem of determining whether a given CNF instance admits a solution was the first to be shown NP-Complete [8]. Binary CNF instances are polynomial [18] but allowing even just 3 literals in each clause (3-SAT) is known to be NP-Complete [13]. Since its inception in Karp's 21 NP-Complete problems, 3-SAT has in fact been a popular problem to reduce from in NP-hardness proofs. Note however that both the Balanced SAT algorithm and our own No-Triangle CNF generator can be used to generate CNF instances of any arity.

The intuition behind our method is to minimize the amount of similarities between clauses. We present a few concepts to help structure this idea. We start by the notion of repeated pairs of variables, which is also used by Balanced SAT.

Definition 2.2. Let I be a CNF instance. Let v and v' be two variables of I. We say that v and v' form a *repeated pair* if they occur together in at least two different clauses of I, regardless of the polarity of their literals.

To benefit from several Graph Theory properties, we view a CNF instance as a graph.

Definition 2.3. Let I be a CNF instance. The *constraint graph* of I is the graph G such that the vertices of G are the variables of I and the edges of G are the pairs of variables of I that occur together in a same clause, regardless of the polarity of their literals.

While a repeated pair of variables forms an edge in the constraint graph, not all constraint graph edges are repeated pairs.

Our No-Triangle CNF algorithm does not just study pairwise relations. It goes one step further and also looks at three-sided transitive structures.

Definition 2.4. Let I be a CNF instance and let v_1, v_2, and v_3 be three variables of I. We say that v_1, v_2, and v_3 form a *constraint triangle* if the three pairs $\langle v_1, v_2 \rangle$, $\langle v_1, v_3 \rangle$, and $\langle v_2, v_3 \rangle$ are edges in the constraint graph of I. If exactly two out of these three pairs are edges in the constraint graph of I, we instead say that v_1, v_2, and v_3 form an *incomplete constraint triangle*.

Note that it is possible for three variables to form a constraint triangle even if no single clause contains all three of them. The notion of constraint triangle is related to the cluster coefficient graph metric, one of the measures characterizing small-world networks [21].

2.2 Balanced SAT

When k, n, and m are the desired arity, number of variables, and number of clauses respectively, the Balanced SAT algorithm [19] creates $\frac{k\times m}{n}$ rows of variables, where each row contains every variable exactly once (except the last row if $k \times m$ is not a multiple of n). The variables are then sorted within each row by a greedy heuristic that picks the variable that minimizes the number of variable pairs in the current clause that are already edges in the constraint graph. Finally, the polarities are assigned randomly for the first occurrence of each variable, and alternatively for the remaining occurrences.

2.3 No-Triangle CNF

Considering only repeated pairs for discriminating between variables can still leave several potential candidates, and Balanced SAT has no choice but to pick one of them randomly. No-Triangle CNF introduces an additional tie-breaker: the number of constraint triangles formed by adding the variable considered to the current clause. This corresponds to Lines 10-15 and 20 in Algorithm 1. While seemingly only a minor alteration, we shall show in the next section that No-Triangle CNF instances behave very differently than Balanced SAT ones.

When trying to generate hard instances, we avoid constraint triangles and encourage incomplete constraint triangles. This mirrors results in Constraint Satisfaction Problems, where it has been shown that the absence of patterns similar to incomplete constraint triangles fulfills the Joint-Winner Property and constitutes a tractable class, while merely forbidding complete constraint triangles still yields an NP-Complete complexity class [9].

3 EXPERIMENTAL RESULTS

We empirically compared four different ways to generate k-CNF instances with n variables and m clauses. The first is Random CNF, where every literal in each clause is randomly picked from the $2n$ possible choices, with no influence from previous picks, with the exception of not allowing the exact same clause twice. This construction is equivalent to a known method for creating CNF instances [1]. The second algorithm is q-Planted SAT [12], which plants a solution in a random CNF instance and tries to hide it by altering the polarity of the literals according to the parameter q. We set the values of q to the ones that for a given k balance the number of positive and negative literals, as computed in a previous SAT competition [4]. The third algorithm is Balanced SAT, and the last one is our No-Triangle CNF.

We tested the four algorithms on 3-CNF instances of sizes $n = 175$, $n = 200$, and $n = 225$, with the results presented in Figure 1, 2, and 3 respectively. To capture the most interesting instances, that is the ones around the peak of difficulty, we varied the number of clauses from $m = 3n$ to $m = 5n$, with a step of 10. The X axis represents the ratio r equal to the number of clauses divided by the number of variables, while the Y axis represents CPU time. Each point in the Figures represents the average of 20 instances. Each instance was solved by four different solvers: Lingeling 18.05 [6], CaDiCal 18 [6], Glucose 4.1 [3], and MapleLCMDistChronoBT [17]. As is the custom in SAT competitions, only the lowest time of the four was kept for each individual instance. Experiments were conducted on a Dell PowerEdge R410 with an Intel Xeon E5620 processor.

The general behavior is the same for all three sizes: the peak of difficulty for No-Triangle CNF is about as tall as the one for Balanced SAT and occurs at a very different ratio (precise results can be found in Table 1). In fact, for 3-CNF instances with at least four times as many clauses as variables,

```
Data: Three integers $k$, $n$, and $m$.
Result: A $k$-CNF instance with $n$ variables $v_1, v_2, \ldots, v_n$ and $m$ clauses $C_1, C_2, \ldots, C_m$.
1  Start with $I$, a CNF instance with $m$ empty clauses $C_1, C_2, \ldots, C_m$;
2  for $i \leftarrow 1$ to $m$ do
3      for $j \leftarrow 1$ to $k$ do
4          Pick the variable $v$ that occurs the fewest in $I$ so far;
5          if several variables are tied then
6              for each tied variable do
7                  Compute the number of repeated pairs added by introducing the variable in $C_i$;
8              end
9              Pick the variable $v$ with the smallest number;
10             if several variables remain tied then
11                 for each tied variable do
12                     Compute the number of constraint triangles added by introducing the
                         variable in $C_i$;
13                 end
14                 Pick the variable $v$ with the smallest number;
15             end
16         end
17         Add the literal $v$ to $C_i$;
18         Increment the number of occurrences of $v$ by 1;
19         Update the database of variable pairs present in $I$;
20         Update the database of incomplete constraint triangles present in $I$;
21     end
22 end
23 for $i \leftarrow 1$ to $n$ do
24     With probability $\frac{1}{2}$, change the first occurrence of $v_i$ in $I$ to a negative literal;
25     Set each subsequent occurrence of $v_i$ to the negation of the previous occurrence;
26 end
27 return $I$;
```

Algorithm 1: No-Triangle CNF generator.

No-Triangle CNF instances are for all three sizes one order of magnitude harder then Balanced SAT instances.

It is important to note that while the time needed to determine whether a given CNF instance admits a solution is solver dependent (hence why we used four different solvers), the answer itself is not. In other words, while the precise height of the peak of difficulty for a particular solver depends on a number of parameters, its location will always be at the transition between satisfiability and unsatisfiablity [16]. To illustrate this phenomenon, we show in Figure 4 where exactly the phase transition takes place for Balanced and No-Triangle 3-CNF instances. This corroborates what we observed in the previous Figures and shows that the differences in behavior between the two methods come from the instances, not from the solvers.

We suspect that as the number of variables increases, the behavior of No-Triangle CNF instances diverges more and more from the one of Balanced SAT instances. Indeed, the maximum possible number of edges in a graph is quadratic in the number of vertices, but because the arity k of the

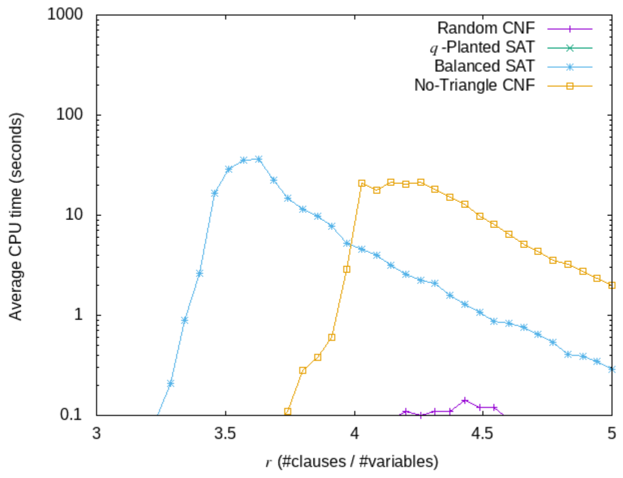

Fig. 1. Instance difficulty for 3-CNF instances with 175 variables.

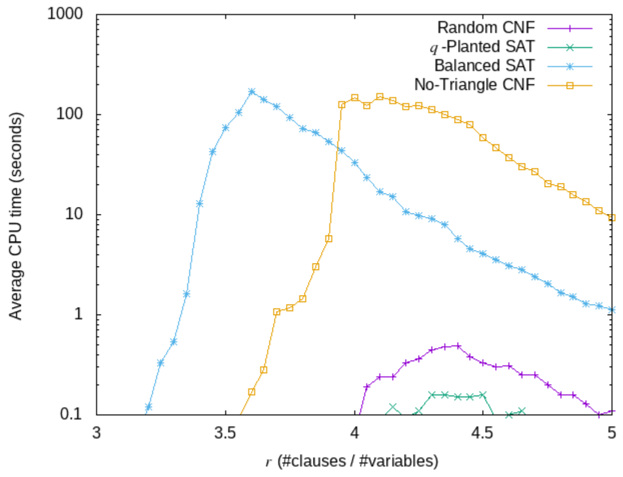

Fig. 2. Instance difficulty for 3-CNF instances with 200 variables.

instances is fixed, the number of actually occurring variable pairs is linear (equal to $3m$ for $k = 3$). Therefore it is easier to avoid repeated pairs for larger values of n, and the additional tie-breaker from No-Triangle CNF will be used more often. This would explain why in our experiments the peak of difficulty for No-Triangle CNF seems to grow faster as n increases than the one for Balanced SAT.

Since we designed No-Triangle CNF to work for any arity, we also experimented on 4-CNF instances. CPU time average plots for $k = 4$ and $n = 100$ can be found in Figure 5. The comparison of phase transitions for this configuration is in Figure 6 and the exact values for the peaks of difficulty are indicated, along with the ones for 3-CNF instances, in Table 1. We also present in Figure 7 the results of experiments on 4-CNF instances with 120 variables, but since the runtimes were prohibitely long for this size we imposed a timeout of 3600 seconds (one hour) for each

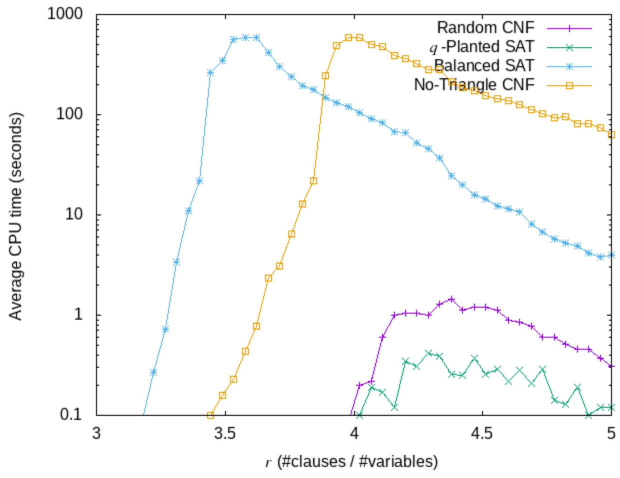

Fig. 3. Instance difficulty for 3-CNF instances with 225 variables.

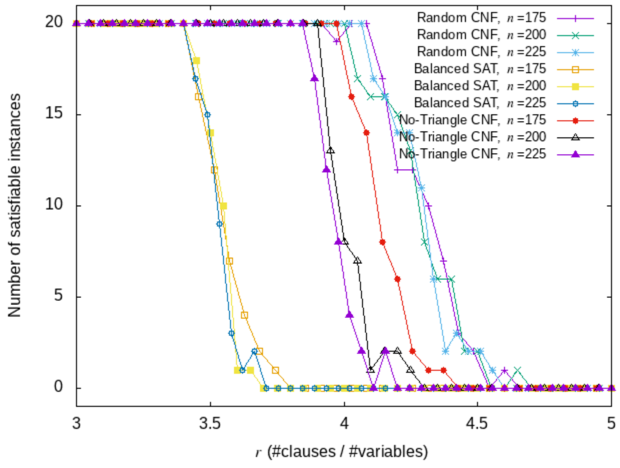

Fig. 4. Phase transitions for 3-CNF instances.

individual instance. The main properties observed in the 3-CNF tests are conserved in the 4-CNF experiments: difficult No-Triangle CNF instances are at least as hard as difficult Balanced SAT instances, and appear in a different constrainedness region. The only oddity is that while in 3-CNF the peak of difficulty for No-Triangle CNF occurs *after* the one for Balanced SAT, in 4-CNF the No-Triangle CNF peak occurs *before* the Balanced SAT one.

As a concrete example of how No-Triangle CNF can find difficult instances in constrainedness regions where other generators only see easy instances, we present in Table 2, for each combination of k and n featured in our experiments, detailed results for one particular number of clauses. So for $k = 3$ and $n = 175$, there exists a number of clauses, namely $m = 775$, for which the average No-Triangle CNF instance is ten times as hard to solve as the average Balanced SAT instance. As n increases, we can find a value of m that improves further this ratio: for $k = 3$, $n = 225$, and

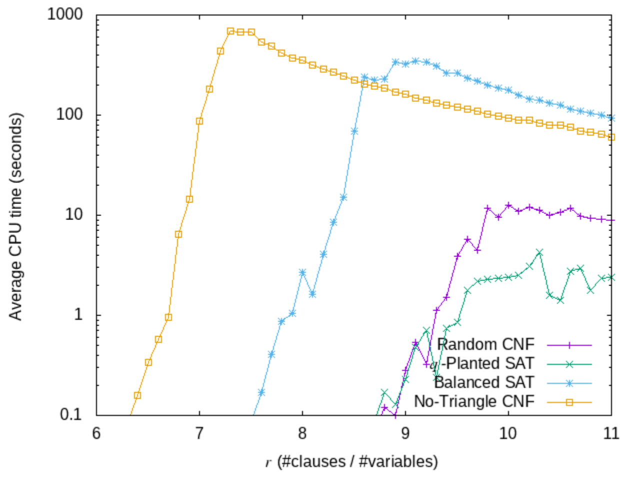

Fig. 5. Instance difficulty for 4-CNF instances with 100 variables.

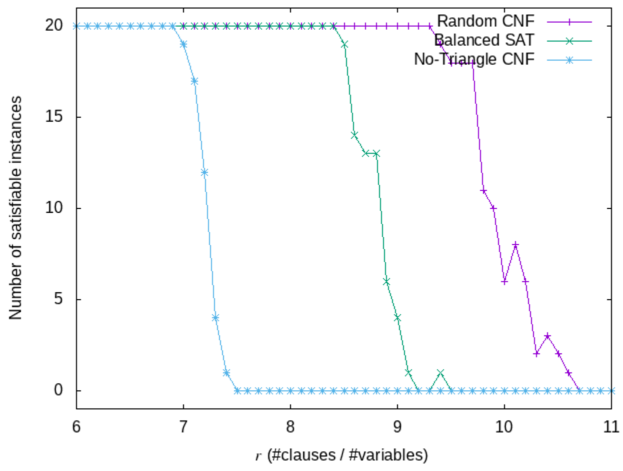

Fig. 6. Phase transitions for 4-CNF instances with 100 variables.

$m = 1085$, the average No-Triangle CNF instance is eighteen times as hard to solve as the average Balanced SAT instance. Since these are averages, it is not guaranteed that No-Triangle CNF will always generate an instance that difficult. However, perhaps more impressively, even the *easiest* No-Triangle CNF instance for $k = 3$, $n = 225$, and $m = 1085$ is still more than ten times as difficult to solve as the *hardest* Balanced SAT instance for the same parameters.

From our 4-CNF experiments, not only can we also pick a value of m that exhibits a clear advantage in favor of No-Triangle CNF, but the difference is even starker. For $n = 100$ and $m = 750$, the average No-Triangle CNF instance is more than eight thousand times as hard to solve as the average Balanced SAT instance. And at the same number of clauses, the easiest No-Triangle CNF instance among the twenty that we generated was still more than fifteen hundred times as difficult to solve as the hardest instance among the twenty that Balanced SAT found. In fact, that easiest

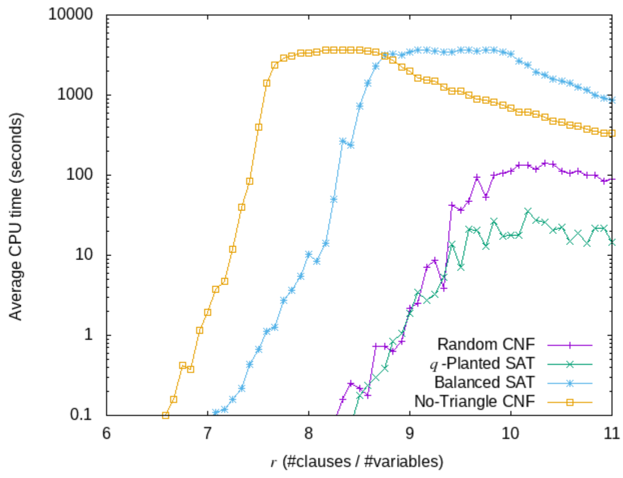

Fig. 7. Instance difficulty for 4-CNF instances with 120 variables.

Table 1. Comparison of the peaks reached by the generators.

		Random CNF		q-Planted SAT		Balanced SAT		No-Triangle CNF	
		m (r)	CPU time (sec)	m (r)	CPU time (sec)	m (r)	CPU time (sec)	m (r)	CPU time (sec)
$k = 3$ $n = 175$	Average	775 (4.4)	.14	805 (4.6)	.04	635 (3.6)	**36.22**	745 (4.3)	21.50
	Median	775 (4.4)	.13	805 (4.6)	.04	635 (3.6)	**44.02**	725 (4.1)	27.03
$k = 3$ $n = 200$	Average	880 (4.4)	.49	900 (4.5)	.16	720 (3.6)	**170.00**	820 (4.1)	151.34
	Median	870 (4.4)	.48	900 (4.5)	.12	720 (3.6)	178.33	800 (4.0)	**212.18**
$k = 3$ $n = 225$	Average	985 (4.4)	1.43	965 (4.3)	.42	815 (3.6)	**592.82**	905 (4.0)	588.66
	Median	985 (4.4)	1.29	965 (4.3)	.34	795 (3.5)	706.74	895 (4.0)	**758.50**
$k = 4$ $n = 100$	Average	1000 (10.0)	12.73	1030 (10.3)	4.22	910 (9.1)	344.70	730 (7.3)	**696.98**
	Median	980 (9.8)	15.09	1030 (10.3)	3.25	890 (8.9)	434.43	730 (7.3)	**807.04**

No-Triangle CNF instance took more time to solve than the average instance at the *peak of difficulty* for Balanced SAT with $k = 4$ and $n = 100$ (see Table 1).

Of course there also exist values for the number of clauses for which Balanced SAT beats No-Triangle CNF. We do not claim that Balanced SAT is made redundant by No-Triangle CNF. What we are claiming is that our algorithm succeeds in parts of the constrainedness map where existing hard instance generators fail.

Table 2. Examples of significant hardness differences between Balanced SAT and No-Triangle CNF. (TO) denotes that the timeout limit was reached.

Size	#clauses	Balanced SAT		No-Triangle CNF	
		CPU time (seconds)		CPU time (seconds)	
		Average	Highest	Lowest	Average
$k = 3, n = 175$	775	1.29	1.70	7.09	12.89
$k = 3, n = 200$	870	7.94	12.33	86.11	98.98
$k = 3, n = 225$	1085	5.29	7.42	76.55	94.61
$k = 4, n = 100$	750	.08	.33	519.98	671.72
$k = 4, n = 120$	980	14.20	48.20	3600.00 (TO)	3600.00 (TO)

Table 3. Solver score by instance size. For each generator/size combination, the number of points scored by a solver indicates the number of instances for which that solver was the fastest. Scores can be non-integer because points are shared in case of ties.

Generator	(k, n)	Lingeling 18.05	CaDiCal 18	Glucose 4.1	MapleLCMDist ChronoBT
Random CNF	(3,175)	26.66	189.66	**413.66**	90.00
	(3,200)	81.75	247.91	**377.91**	112.41
	(3,225)	175.25	240.41	**400.41**	103.91
	(4,100)	155.08	229.25	**566.08**	69.58
q-Planted SAT	(3,175)	24.91	234.91	**374.75**	85.41
	(3,200)	66.91	288.41	**367.41**	97.25
	(3,225)	128.33	315.33	**362.33**	114.00
	(4,100)	208.16	300.50	**439.00**	72.33
Balanced SAT	(3,175)	109.50	110.00	**489.00**	11.50
	(3,200)	125.00	78.16	**480.66**	136.16
	(3,225)	90.16	72.66	**433.66**	323.50
	(4,100)	79.00	173.16	274.16	**493.66**
No-Triangle CNF	(3,175)	90.00	141.00	**465.00**	24.00
	(3,200)	69.08	150.08	**390.25**	210.58
	(3,225)	97.16	127.33	206.33	**489.16**
	(4,100)	59.50	80.50	132.00	**748.00**

We display in Tables 3 and 4 some statistics of interest pertaining to the solvers used. For each of the instances we generated, we added 1 point to the score of the solver that solved the instance in the lowest time. If two (respectively three, four) solvers shared the lowest time for one particular instance, they each got half (respectively a third, a fourth) of a point on that instance. We did not include the 4-CNF instances with 120 variables, as the presence of instances on which all solvers time out would have skewed the results.

Table 3 highlights a difference in performance between Glucose and Maple for large 3-CNF instances. Glucose dominates Maple on Balanced SAT instances with $k = 3$ and $n = 225$, while the reverse is true for No-Triangle CNF instances with the same parameters. In general Maple seems to perform better on difficult instances. This is confirmed by Table 4 which shows that Maple was the fastest solver on all 155 instances that required at least ten minutes to be solved. It is interesting to note that more than two thirds of these instances were generated by No-Triangle CNF. No-Triangle

Table 4. Solver score by instance difficulty. For each generator/difficulty combination, the number of points scored by a solver indicates the number of instances for which that solver was the fastest. Scores can be non-integer because points are shared in case of ties.

Generator	CPU time (seconds)	Lingeling 18.05	CaDiCal 18	Glucose 4.1	MapleLCMDist ChronoBT
Random CNF	<30	438.75	907.25	**1758.08**	375.91
	≥30	0	0	0	0
q-Planted SAT	<30	428.33	1139.16	**1543.50**	369.00
	≥30	0	0	0	0
Balanced SAT	<30	394.66	410.00	**1487.50**	71.83
	30-599	9.00	24.00	190.00	**844.00**
	≥600	0	0	0	**49.00**
No-Triangle CNF	<30	297.75	472.91	**1028.58**	86.75
	30-599	18.00	26.00	165.00	**1279.00**
	≥600	0	0	0	**106.00**

CNF is also responsible for 58% of the moderately hard instances, the ones requiring between thirty seconds and ten minutes to solve.

4 CONCLUSION

We have introduced No-Triangle CNF, an algorithm that can generate CNF instances with any arity, number of variables, and/or number of clauses. In particular No-Triangle CNF can build very difficult instances in constrainedness regions where no hard instance had been previously generated. For some combinations of the aforementioned parameters, the easiest instance created by No-Triangle CNF is still more than 1500 times as hard to solve as the most difficult instance returned by the state-of-the-art Balanced SAT generator.

Our results show that No-Triangle CNF is useful at tightness values where other instance generators fail to provide difficult instances. It can also be combined with other algorithms like Balanced SAT to form an instance generator with a large, comprehensive scope. Additionally, we believe that future work on constraint triangles could help solver heuristics for instances with a low constraint graph cluster coefficient.

ACKNOWLEDGMENTS

This material is based upon works supported by the Science Foundation Ireland under Grant No. 12/RC/2289-P2 which is co-funded under the European Regional Development Fund.

We wish to thank the anonymous reviewers for their comments and suggestions, which helped us improve our paper.

REFERENCES

[1] Dimitris Achlioptas. 2009. Random Satisfiability. In *Handbook of Satisfiability*, Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (Eds.). Frontiers in Artificial Intelligence and Applications, Vol. 185. IOS Press, 245–270. https://doi.org/10.3233/978-1-58603-929-5-245

[2] Giovanni Amendola, Francesco Ricca, and Miroslaw Truszczynski. 2017. Generating Hard Random Boolean Formulas and Disjunctive Logic Programs. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, Carles Sierra (Ed.). ijcai.org, 532–538. https://doi.org/10.24963/ijcai.2017/75

[3] Gilles Audemard and Laurent Simon. 2018. On the Glucose SAT Solver. *International Journal on Artificial Intelligence Tools* 27, 1 (2018), 1–25. https://doi.org/10.1142/S0218213018400018

[4] Adrian Balint. 2018. Random k-SAT q-planted solutions.. In *Proc. of SAT Competition 2018 – Solver and Benchmark Descriptions (Department of Computer Science Series of Publications B)*, Marijn Heule, Matti Järvisalo, and Martin Suda (Eds.), Vol. B-2018-1. University of Helsinki, 64.

[5] Tomás Balyo and Lukás Chrpa. 2018. Using Algorithm Configuration Tools to Generate Hard SAT Benchmarks. In *Proceedings of the Eleventh International Symposium on Combinatorial Search, SOCS 2018, Stockholm, Sweden - 14-15 July 2018*, Vadim Bulitko and Sabine Storandt (Eds.). AAAI Press, 133–137. https://aaai.org/ocs/index.php/SOCS/SOCS18/paper/view/17952

[6] Armin Biere. 2018. CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2018. In *Proc. of SAT Competition 2018 – Solver and Benchmark Descriptions (Department of Computer Science Series of Publications B)*, Marijn Heule, Matti Järvisalo, and Martin Suda (Eds.), Vol. B-2018-1. University of Helsinki, 13–14.

[7] Peter C. Cheeseman, Bob Kanefsky, and William M. Taylor. 1991. Where the Really Hard Problems Are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*, John Mylopoulos and Raymond Reiter (Eds.). Morgan Kaufmann, 331–340. http://ijcai.org/Proceedings/91-1/Papers/052.pdf

[8] Stephen A. Cook. 1971. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman (Eds.). ACM, 151–158. https://doi.org/10.1145/800157.805047

[9] Martin C. Cooper and Stanislav Zivny. 2011. Tractable Triangles. In *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings (Lecture Notes in Computer Science)*, Jimmy Ho-Man Lee (Ed.), Vol. 6876. Springer, 195–209. https://doi.org/10.1007/978-3-642-23786-7_17

[10] James M. Crawford and Larry D. Auton. 1996. Experimental Results on the Crossover Point in Random 3-SAT. *Artif. Intell.* 81, 1-2 (1996), 31–57. https://doi.org/10.1016/0004-3702(95)00046-1

[11] Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. 2012. The International SAT Solver Competitions. *AI Magazine* 33, 1 (2012). http://www.aaai.org/ojs/index.php/aimagazine/article/view/2395

[12] Haixia Jia, Cristopher Moore, and Doug Strain. 2007. Generating Hard Satisfiable Formulas by Hiding Solutions Deceptively. *J. Artif. Intell. Res.* 28 (2007), 107–118. https://doi.org/10.1613/jair.2039

[13] Richard M. Karp. 1972. Reducibility Among Combinatorial Problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA (The IBM Research Symposia Series)*, Raymond E. Miller and James W. Thatcher (Eds.). Plenum Press, New York, 85–103. http://www.cs.berkeley.edu/%7Eluca/cs172/karp.pdf

[14] Donald E. Knuth. 2015. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability* (1st ed.). Addison-Wesley Professional.

[15] Stephan Mertens, Marc Mézard, and Riccardo Zecchina. 2006. Threshold values of random K-SAT from the cavity method. *Random Struct. Algorithms* 28, 3 (2006), 340–373. https://doi.org/10.1002/rsa.20090

[16] David G. Mitchell, Bart Selman, and Hector J. Levesque. 1992. Hard and Easy Distributions of SAT Problems. In *Proceedings of the 10th National Conference on Artificial Intelligence, San Jose, CA, USA, July 12-16, 1992*, William R. Swartout (Ed.). AAAI Press / The MIT Press, 459–465. http://www.aaai.org/Library/AAAI/1992/aaai92-071.php

[17] Alexander Nadel and Vadim Ryvchin. 2018. Chronological Backtracking. In *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings (Lecture Notes in Computer Science)*, Olaf Beyersdorff and Christoph M. Wintersteiger (Eds.), Vol. 10929. Springer, 111–121. https://doi.org/10.1007/978-3-319-94144-8_7

[18] Thomas J. Schaefer. 1978. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho (Eds.). ACM, 216–226. https://doi.org/10.1145/800133.804350

[19] Ivor Spence. 2017. Balanced random SAT benchmarks. In *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions (Department of Computer Science Series of Publications B)*, Tomáš Balyo, Marijn Heule, and Matti Järvisalo (Eds.), Vol. B-2017-1. University of Helsinki, 53–54.

[20] Ivor T. A. Spence. 2015. Weakening Cardinality Constraints Creates Harder Satisfiability Benchmarks. *ACM Journal of Experimental Algorithmics* 20 (2015), 1.4:1–1.4:14. https://doi.org/10.1145/2746239

[21] Duncan J. Watts and Steven H. Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature* 393 (1998), 440–442. https://doi.org/10.1038/30918